

A QR-based solver for rank structured matrices

Steven Delvaux Marc Van Barel

Report TW454, March 2006



Katholieke Universiteit Leuven
Department of Computer Science
Celestijnenlaan 200A – B-3001 Heverlee (Belgium)

A QR-based solver for rank structured matrices

Steven Delvaux *Marc Van Barel*

Report TW454, March 2006

Department of Computer Science, K.U.Leuven

Abstract

In this paper we show how to compute the QR-factorization of a rank structured matrix in an efficient way, using the Givens-weight representation which we introduced in an earlier paper. We also show how the QR-factorization can be used as a preprocessing step for the solution of linear systems. The performance of this scheme will be demonstrated by the results of some numerical experiments.

Keywords : rank structured matrix, Givens-weight representation, QR-factorization, structure inheritance, linear system solution.

AMS(MOS) Classification : Primary : 65F05, Secondary : 65F25, 15A03.

A QR-BASED SOLVER FOR RANK STRUCTURED MATRICES

STEVEN DELVAUX *, MARC VAN BAREL *

Abstract. In this paper we show how to compute the QR-factorization of a rank structured matrix in an efficient way, using the Givens-weight representation which we introduced in an earlier paper. We also show how the QR-factorization can be used as a preprocessing step for the solution of linear systems. The performance of this scheme will be demonstrated by the results of some numerical experiments.

Keywords: rank structured matrix, Givens-weight representation, QR-factorization, structure inheritance, linear system solution.

AMS subject classifications: 65F05, 65F25, 15A03

1. Introduction. In this paper we describe how for a rank structured matrix with precomputed Givens-weight representation, one can efficiently compute its QR-factorization and subsequently use this factorization for the solution of linear systems.

A matrix will be called *rank structured* if the ranks of certain submatrices starting from its bottom left corner, as well as the ranks of certain submatrices starting from its top right corner, are small compared to the matrix size.

Rank structured matrices first appeared in the literature in the form of *uv-representations*, which can be used in case the block lower triangular part of the matrix coincides with that of a full rank- r matrix, and similarly for the block upper triangular part: see e.g. [11]. More recently they appeared in the literature as (block) *quasiseparable* representations: see e.g. [6, 7]. The reader may consult the description of these two representations in [4] for more information.

In the paper [4], we introduced yet another type of representation for rank structured matrices, which we called the *unitary-weight representation*. We showed how this representation is theoretically equivalent with the block quasiseparable representations in *input normal form* described in the book [6]. Nevertheless, we note that we are not aware of any systematic treatment of the use of the latter representations in the literature.

In [4], it was also shown how the unitary-weight representation can be further specified to the *Givens-weight representation*, hereby generalizing from [13]. This representation is more canonically determined in the sense that it allows to obtain efficient representations consisting of $O((r+s)n)$ parameters, for *any* distribution of the low rank blocks, where n is the matrix size, r is some measure for the average rank index of the rank structure, and s is some measure for the bandwidth of the unstructured matrix part around the main diagonal.

In the present paper, we consider the problem of solving linear systems with rank structured coefficient matrix. Many methods for doing this have already been

*Department of Computer Science, Katholieke Universiteit Leuven, Celestijnenlaan 200A, B-3001 Leuven (Heverlee), Belgium. email: {Steven.Delvaux,Marc.VanBarel} @cs.kuleuven.ac.be.

The research was partially supported by the Research Council K.U.Leuven, project OT/05/40 (Large rank structured matrix computations), Center of Excellence: Optimization in Engineering, by the Fund for Scientific Research–Flanders (Belgium), G.0455.0 (RHPH: Riemann-Hilbert problems, random matrices and Padé-Hermite approximation), G.0423.05 (RAM: Rational modelling: optimal conditioning and stable algorithms), and by the Belgian Programme on Interuniversity Poles of Attraction, initiated by the Belgian State, Prime Minister’s Office for Science, Technology and Culture, project IUAP V-22 (Dynamical Systems and Control: Computation, Identification & Modelling). The scientific responsibility rests with the authors.

described in the literature. At the risk of making a too crude distinction, these methods can be divided in at least three categories.

A first type of solvers is based on the Sherman-Morrison formula and its generalizations: see e.g. [8, Section 7] and [10, Section 4] for the case of uv -representable matrices with low rank blocks lying just below the main diagonal.

A second type of solvers is based on LU-factorizations and Gaussian operations without pivoting, a fact which is often revealed by the condition that the matrix must be strongly nonsingular. See e.g. [8, Section 5] for the case of block quasiseparable matrices with low rank blocks lying just below the main diagonal, and see [10, Section 5] for an algorithm in the uv -representable case.

Finally, a third type of solvers is based on QR- or URV-factorizations. The first such algorithm was the URV-decomposition solver for block quasiseparable matrices reported in the book [6, Chapter 7]. More efficient versions of this algorithm were then first obtained in [9] using the QR-decomposition (see also [12]), and more recently in [2, 1] using a URV-decomposition in case where the low rank blocks are situated just below the main diagonal.

In the present paper, we follow the solution strategy of [9] by developing a linear system solver that is based on a preliminary QR-factorization. The algorithm will be expressed in terms of the Givens-weight representation. The computation of the QR-factorization will require about $O((r+s)^2n)$ operations. This QR-factorization can then be used for the efficient solution of a linear system, the latter requiring only $O((r+s)n)$ operations anymore.

The algorithm in this paper will be able to capture *any* rank structure, irrespective of the position of the low rank blocks w.r.t. each other and to the main diagonal. The only restriction is that the low rank blocks in the block lower triangular part, may not overlap with those in the block upper triangular part.

The remainder of this paper is organized as follows. In Section 2 we recall the basic ideas of the Givens-weight representation from [4]. Section 3 considers the QR-factorization of a rank structured matrix. This section contains both a theoretical part concerning structure inheritance by the Q- and R-factors of the QR-factorization, as well as a practical part concerning the algorithmic exploitation of these inheritance results. Section 4 deals with the linear system solver. Finally, Section 5 reports on the results of some numerical experiments.

2. Givens-weight representations. In this section we review the basic ideas of the Givens-weight representation from [4].

First we define the class of rank structured matrices.

DEFINITION 1. (See [3]:) *We define a pure rank structure \mathcal{R} on $\mathbb{C}^{m \times n}$ as a collection of so-called pure structure blocks $\mathcal{R} = \{\mathcal{B}_k\}_k$. Each pure structure block \mathcal{B}_k is characterized as a 3-tuple*

$$\mathcal{B}_k = (i_k, j_k, r_k),$$

where i_k is the row index, j_k the column index, r_k the rank upper bound. We say a matrix $A \in \mathbb{C}^{m \times n}$ to satisfy the pure rank structure \mathcal{R} if for each k ,

$$\text{Rank } A(i_k : m, 1 : j_k) \leq r_k.$$

The above definition uses the word *pure* to distinguish from the more general rank structures which were handled in [3]. Since these more general structures do not

occur in the present paper, we will simplify notation by just dropping the word *pure* everywhere from the notation.

Note that by definition, all structure blocks have to start from the lower left matrix corner. An example of a rank structure is shown in Figure 2.1.

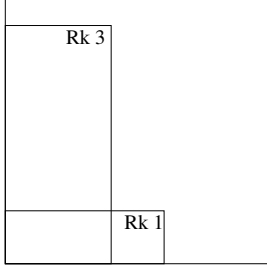


FIGURE 2.1. Example of a rank structure with two structure blocks \mathcal{B}_1 and \mathcal{B}_2 . The notation ‘Rk r ’ denotes that the structure block is of rank at most r .

In practice, it often happens that also the block *upper* triangular part is rank structured, i.e., that also the matrix A^T satisfies rank structure in the sense of Definition 1. By abuse of notation, we will indiscriminately use the term *rank structure* also in this case.

We will assume in what follows that we are working with a rank structure \mathcal{R} for which there are no structure blocks that are ‘contained’ in each other, i.e., for which the structure blocks \mathcal{B}_k can be ordered such that both their row and column indices i_k and j_k increase in a strictly monotonic way. (Actually, these ‘internal’ structure blocks are not completely useless, in the sense that they lead to an additional sparsity pattern in the Givens-weight representation, but we will not be concerned about this here.)

We can then define unitary-weight representations.

DEFINITION 2. (*Unitary-weight representation. See [4]:*) Let $A \in \mathbb{C}^{m \times n}$ be a matrix satisfying a rank structure $\mathcal{R} = \{\mathcal{B}_k\}_{k=1}^K$, where the structure blocks are ordered from top left to bottom right. A unitary-weight representation of the matrix A according to the structure \mathcal{R} consists of a pair $(\{U_k\}_{k=1}^K, W)$. Here the U_k , $k = K, \dots, 1$ form a sequence of unitary transformations proceeding from bottom to top of the matrix, and serving to create zeros in the subsequent Rk r_k structure blocks \mathcal{B}_k except for their top r_k rows. On the other hand, the matrix $W \in \mathbb{C}^{m \times n}$ is called the weight matrix, and it contains the blocks of elements obtained at the top border of the rank structure at the moment just after applying U_k . See Figure 2.2.

The basic idea of this definition is to compress the given rank structured matrix by means of subsequent unitary transformations, hereby proceeding from bottom to top of the matrix, and storing each time the elements just before they reach the top border of the rank structure.

Note that this definition leads to an *internal* representation of the rank structure, in the sense that it involves no information about the matrix part lying outside the reach of the structure blocks. Moreover, it implies that each unitary operation U_k has a certain *action radius* in the sense that it acts only on a limited number of columns. This action radius is a monotonically decreasing function when the U_k proceed from bottom to top of the matrix.

If a unitary-weight representation of a matrix is given, we can restore the full matrix by *spreading out* the representation. This means that we gradually consider

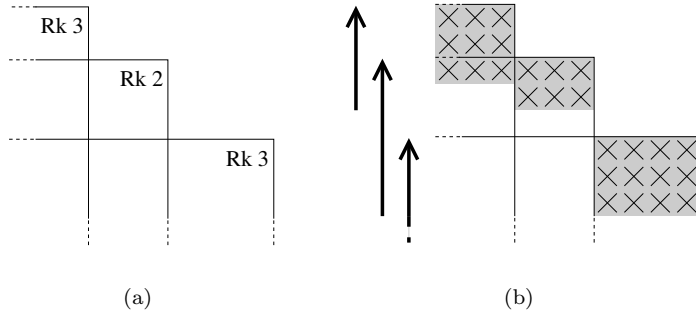


FIGURE 2.2. For the rank structure in the left picture, the right figure shows a schematic picture of the unitary-weight representation.

the subsequent weight blocks, proceeding from top to bottom of the matrix, and multiply them with the ‘decompressing’ unitary operations U_k^{-1} , each one acting of course only on the columns on the left of its action radius. While this process proceeds from top to bottom of the matrix, we will gradually retrieve the original, full matrix which we started from.

We can now specify from unitary-weight to Givens-weight representations. In what follows, we will use the term *Givens transformation* to denote a unitary operation which differs from the identity matrix only in two subsequent rows i and $i + 1$. This transformation will sometimes be denoted as $G_{i,i+1}$, and the index i will be called the *row index* of the Givens transformation.

First, rather than individual Givens transformations, it will be useful to work with *Givens arrows*: these are defined as collections of subsequent Givens transformations, each of them having row index precisely one more than the previous one. This means that each Givens transformation is situated precisely one position below the previous one: see Figure 2.3.

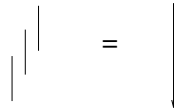


FIGURE 2.3. A Givens-arrow consisting of 3 Givens transformations. Concerning this figure, we recall the reader that we consider each Givens transformation as ‘acting’ on the rows of an (invisible) matrix standing on the right of it, and hence that the Givens transformations in the figure should be evaluated from right to left, hereby explaining the downward direction of the Givens arrow.

The number of Givens transformations of which a Givens arrow consists will be called the *width* of the Givens arrow. Moreover, we define the *top* and the *tail* of the Givens arrow to be the largest and the smallest row index of the Givens transformations of which the Givens arrow consists, respectively. These notions have an obvious graphical interpretation.

Having introduced all these notions, we can now specify from unitary-weight to Givens-weight representations.

DEFINITION 3. (*Givens-weight representation. See [4]:*) Let $A \in \mathbb{C}^{m \times n}$ be a matrix satisfying a rank structure $\mathcal{R} = \{\mathcal{B}_k\}$, where the structure blocks are ordered from

top left to bottom right. A Givens-weight representation of A according to the structure \mathcal{R} is a unitary-weight representation where additionally each unitary component U_k is decomposed into a product of Givens arrows, such that

- each of the Givens arrows has width at most r_k ,
- both the tops and the tails of the subsequent Givens arrows of each U_k are monotonically proceeding upwards. For the tails, we assume that this monotonicity is strict.

See Figure 2.4.

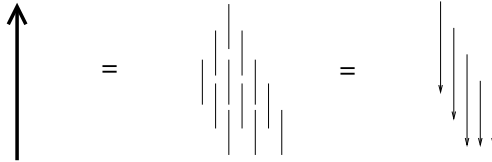


FIGURE 2.4. Suppose that the current structure block is $\text{Rk } 3$, and that the corresponding unitary transformation U_k spans over 6 rows. Then we assume for this unitary transformation a decomposition into a product of Givens arrows of width at most 3.

Of course we should explain why the assumption is made that each Givens arrow in the decomposition of U_k has width at most r_k . But this is a logical assumption: recall that the unitary transformation U_k serves to create zeros in a certain $\text{Rk}(r_k)$ submatrix, except for its top r_k rows, and no matter if we do this by means of a singular value decomposition or by a pivoted QR-factorization or any other unitary operation, this effect can always be realized by a succession of Givens arrows as prescribed.

Note that by decomposing each unitary transformation U_k as specified in Definition 3, we formally obtain a decomposition into a product of *too many* Givens transformations, in the sense that the beginning and trailing Givens transformations of two subsequent unitary transformations U_k may overlap. To avoid such an overlap, some *canonical* Givens-weight representations were described in [4], which do not suffer from this overlap, as well as some general techniques for reducing any Givens-weight representation into canonical form. Nevertheless, the algorithms in this paper will work for *any* Givens-weight representation.

It is easy to see that instead of row operations, one can also build a Givens-weight representation based on *column* operations, and that similar techniques can be used for the structured *upper* triangular part. An example of a Givens-weight representation for a symmetric full rank structured matrix is shown in Figure 2.5.

3. QR-factorization. In this section we describe an algorithm to perform the QR-factorization of a rank structured matrix, assuming that there is given a Givens-weight representation for this matrix. The output of the algorithm consists of the Q- and R-factors of the QR-factorization, where the Q-factor is decomposed as a product of Givens transformations, and where the R-factor has the form of a Givens-weight representation.

Before describing the algorithm, we will start with some theory concerning the structure which we may hope to exploit.

3.1. Some theory. In this subsection we recall and provide some theoretical results concerning the structure inheritance by the Q- and R-factors of a QR-factorization $A = QR$, where A is assumed to be a rank structured matrix.

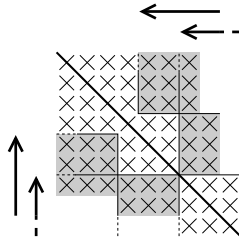


FIGURE 2.5. Schematic picture of a Givens-weight representation for a matrix A having rank structure both in its structured lower and in its structured upper triangular part. The figure shows the weight matrix as well as the unitary transformations of the representation. Note that the matrix is assumed to be symmetric, and that the lower and upper triangular representations are based on row and column operations, respectively. None of these conditions is essential, however.

The inheritance of structure by the Q-factor in terms of Givens transformations was handled in [5]. We considered there the matrix Q^H as a product of Givens transformations ‘acting’ on the matrix A , hereby transforming it into an upper triangular matrix $R = Q^H A$. This process proceeds in two phases.

For the first phase, we recall from Section 2 that for a rank structured matrix A , a sequence of unitary operations can be applied to transform the given $\text{Rk } r_k$ structure blocks of A into blocks of zeros, except for their top r_k rows. This process proceeds from bottom to top of the matrix. We will call this the *preparative phase*.

For the second phase, we note that the resulting matrix A at the end of the preparative phase will be almost zero in its lower triangular part, except for a few non-zero elements around the main diagonal. These remaining non-zero elements can then be annihilated by a sequence of upward pointing Givens arrows, proceeding from top to bottom of the matrix. We will call this the *residual phase*.

We want now to investigate the inheritance of structure by the upper triangular matrix $R = Q^H A$ obtained at the end of the residual phase. It turns out that we have to exclude some pathological cases.

DEFINITION 4. (*Structure implying rank deficiency:*) Let $\mathcal{B} = (i, j, r)$ be a structure block on $\mathbb{C}^{m \times n}$ and denote by $n_{\mathcal{B}} := j - i + 1$ the number of elements cut out by \mathcal{B} on the main diagonal. The structure block is said to imply rank deficiency if $n_{\mathcal{B}}$ is strictly larger than its rank index r .

It can be shown that a structure block implying rank deficiency is equivalent with the structure block *itself* causing a linear dependency between the columns of the underlying matrix. In particular, in the case of square matrices the definition reduces to that of *structure implying singularity* [5]; but we will use Definition 4 for any value of m and n .

In what follows, we will assume that the structure does not imply rank deficiency. This is not really a restriction, since for a structure block implying rank deficiency, one can just remove a few rows or columns from the structure block until it does not imply rank deficiency anymore.

DEFINITION 5. (*Sparsity pattern:*) Let $A \in \mathbb{C}^{m \times n}$ be a matrix satisfying a structure block $\mathcal{B} = (i, j, r)$, not implying rank deficiency. We say a QR-factorization $A = QR$ to satisfy the sparsity pattern induced by the structure block \mathcal{B} if Q^H can be written as a decomposition $Q^H = Q_3^H Q_2^H Q_1^H$, with

- Q_1^H acting on rows i, \dots, m (serving to transform \mathcal{B} into a block of zeros, except for its top r rows),

- Q_2^H acting on rows $1, \dots, i + r - 1$ (serving to create zeros above \mathcal{B}),
- Q_3^H acting on rows $j + 1, \dots, m$ (serving to create zeros on the right of \mathcal{B}).

See Figure 3.1.

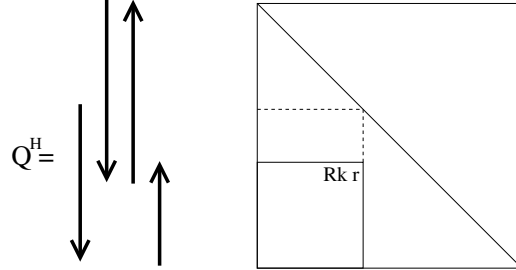


FIGURE 3.1. For the structure block \mathcal{B} shown on the right, the left picture shows the corresponding sparsity pattern of the matrix Q^H w.r.t. this structure block, transforming it into an upper triangular matrix $Q^H A = R$. The picture shows a decomposition into preparative (upward) and residual (downward) Givens transformations.

We should stress that the above definition was formulated from the point of view of a *single* structure block. In practical situations, there will probably be more than one structure block, causing each of the unitary components Q_i^H , $i = 1, 2, 3$ to have an additional decomposition into a sparse product of Givens transformations. To stress this point, we tried to indicate in Figure 3.1 the connection with the preparative and residual Givens transformations.

Since the above definition was formulated from the point of view of a single structure block, it does not reflect the complete sparsity pattern of the preparative and residual unitary operations. Still it will be sufficient to establish the following result.

THEOREM 6. (*Inheritance of structure by the R-factor:*) Let $A \in \mathbb{C}^{m \times n}$ be a matrix having two low rank submatrices $A(I_1, J_1) = \text{Rk } r$ and $A(I_2, J_2) = \text{Rk } s$, where

- I_1 and I_2 form a partition of the index set $\{1, 2, \dots, m\}$,
- $J_1 = \{1, 2, \dots, j_1\}$, for certain j_1 .
- J_2 is arbitrary.

Then for the QR-factorization $A = QR$ it holds that $R(J_1, J_2) = \text{Rk}(r + s)$, see Figure 3.2, at least provided A is square nonsingular.

In case A is a general rectangular matrix, the above property remains valid provided the following holds: assume that P is a row permutation bringing the index sets I_1 and I_2 into the forms $\{1, 2, \dots, i\}$ and $\{i + 1, i + 2, \dots, m\}$ for certain i , as in in Figure 3.3. Note that this permutation turns the given $\text{Rk } r$ block into a ‘real’ structure block situated in the bottom left matrix corner. Then the above theorem remains valid provided the structure block \mathcal{B} does not imply rank deficiency, and provided the QR-factorization $PA = PQR$ satisfies the sparsity pattern induced by \mathcal{B} in the sense of Definition 5.

PROOF. First consider the case where A is nonsingular. First, we claim that the Q-factor must ‘inherit’ each of the $\text{Rk } r$ low rank blocks situated entirely at the left border of A . Indeed, this is evident by using the equation $Q = AR^{-1}$, where the matrix R^{-1} takes linear combinations of the columns of A , hereby only involving

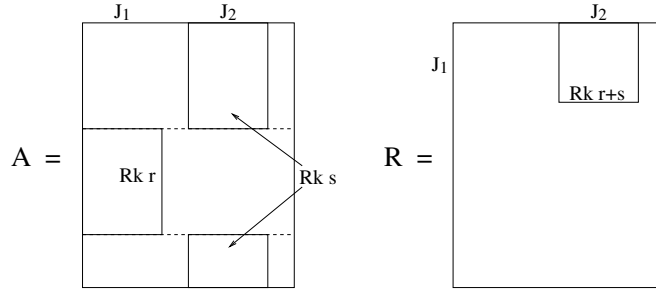


FIGURE 3.2. The figure shows the inheritance of structure by the R-factor for a general example. We note that the index sets I_1 and I_2 must be complementary to each other.

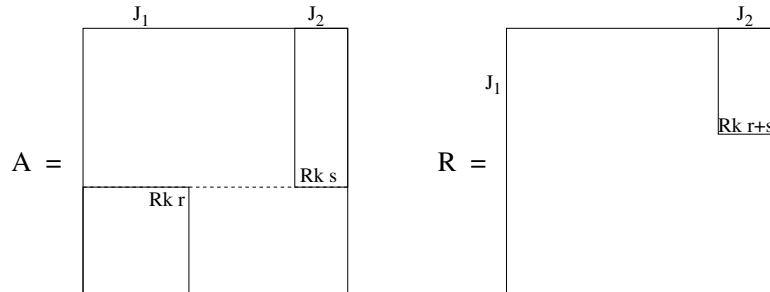


FIGURE 3.3. The figure shows the practical form in which we will exploit the inheritance of structure by the R-factor of Figure 3.2. In particular, it is assumed in this figure and in the sequel that the index set J_2 takes the form $\{j_2, \dots, n\}$ for certain j_2 , since this is the practical form in which the inheritance result will be exploited.

‘previous’ columns, and hence not destroying such low rank blocks (See also [3]). This establishes the inheritance of structure by the Q-factor. Using this, the inheritance of structure by the R-factor follows in an easy way by writing it as the matrix-matrix product $R = Q^H A$: see Figure 3.4.

In case A is a general rectangular matrix, the theorem can be proved by a direct argument in terms of the sparsity pattern of Q^H induced by the Rk r structure block, as is done in Figure 3.5. \square

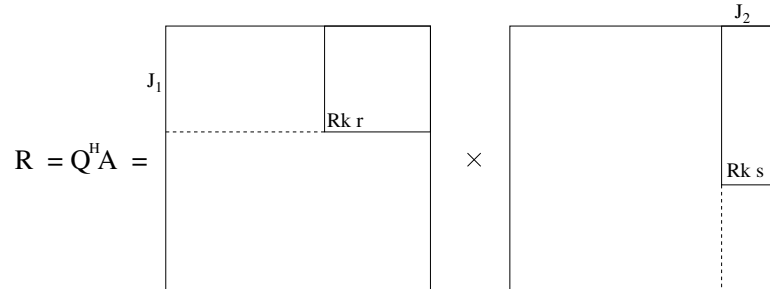


FIGURE 3.4. The figure shows the R-factor of Figure 3.3 in the form $R = Q^H A$, where both factors are rank structured due to the given structure of A , and due to the inheritance of structure by the Q-factor described in the proof of Theorem 6. By the complementarity of the index sets I_1 and I_2 , it is then easy to see that the submatrix $R(J_1, J_2)$ has indeed rank at most $r + s$.

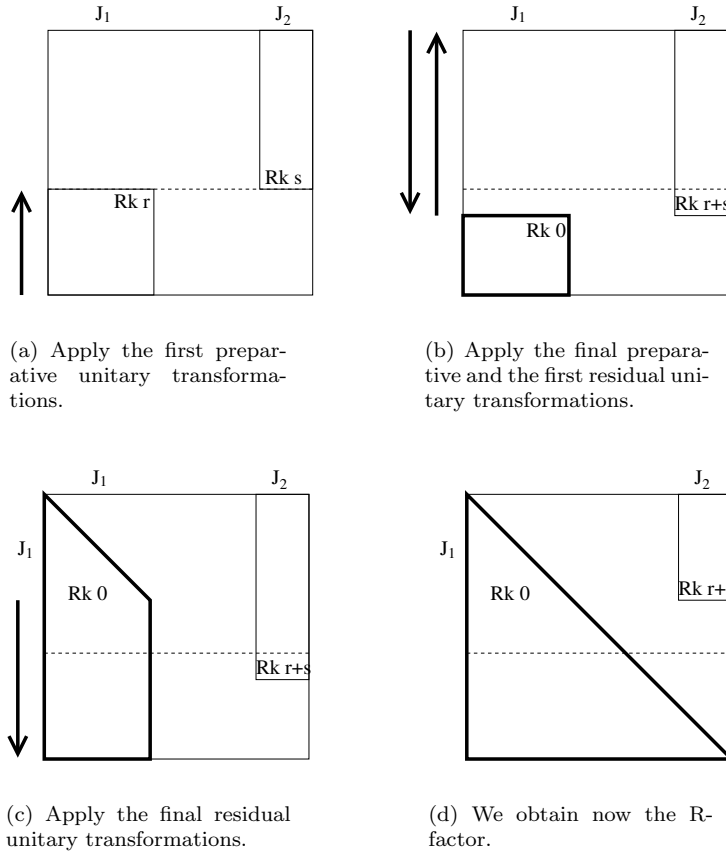


FIGURE 3.5. The figure shows the inheritance of structure by the R-factor of Figure 3.3 using a direct argument in terms of the sparsity pattern of the matrix Q^H .

One may ask where in the above proof the condition is used that the rank structure does not imply rank deficiency. This is done in Figure 3.5(b), where this condition guarantees that the created block of zeros is situated entirely in the strictly lower triangular part of the matrix. The reader might wish to find out what goes wrong if this condition is violated.

Note that the above formulation of structure inheritance by the R-factor has been formulated entirely in terms of index sets. But there exists also a more ‘geometric’ formulation: considering again the low rank block satisfied by the R-factor of the QR-factorization of A in Figure 3.3, we note that this low rank block has exactly the same shape as the original $Rk\ s$ structure block of A , to which are added n_B rows. Here the number $n_B := j - i + 1$ is determined by the $Rk\ r$ structure block, denoting the number of elements cut out by this structure block on the main diagonal, or the ‘distance to the main diagonal’ in case this value is negative. The reader might check the correctness of this formulation.

For the remainder of this section, we turn to a practical exploitation of the above inheritance results for the Q- and R-factors of the QR-factorization.

3.2. Algorithm for the preparative phase. In the next two subsections, we will work under the condition that A is a rank structured matrix for which a Givens-weight representation is available. More precisely, it will be assumed that the structured lower triangular part of A is represented by a *row*-based Givens-weight representation, while the structured upper triangular part is represented by a *column*-based Givens-weight representation. Moreover, the structure blocks of the block lower triangular part are not allowed to overlap with those of the block upper triangular part.

Given these input conditions, we will first describe an algorithm for the first phase of the QR-factorization, the so-called *preparative phase*. We will illustrate the algorithm for a general type of rank structure, a slice of which is shown in Figure 3.6.

The corresponding Givens-weight representation is then shown in Figure 3.7. Note that this figure shows rather a *unitary*-weight than a *Givens*-weight representation, in order not to overload the figure. But the reader should keep in mind that by definition, each unitary component U_k is defined as a sparse Givens product in the style of Figure 2.4.

The application of the preparative phase makes use of the general techniques for updating the Givens-weight representation under the influence of Givens transformations reported in [4], in the form of what we called there a *generalized swapping process*. The process is shown in Figure 3.8.

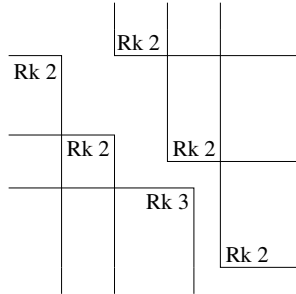


FIGURE 3.6. Starting rank structure.

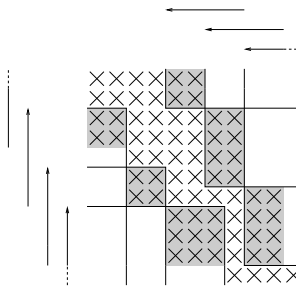


FIGURE 3.7. Starting Givens-weight representation. For clarity of the figure, the Givens transformations are shown grouped together as unitary operations.

Let us comment on this figure. Figure 3.8(a) shows the starting Givens-weight representation. We assume here that the bottommost structure block has already been transformed into a block of zeros, resulting in the fact that its corresponding unitary operation U_k has disappeared from the representation, and that the corresponding

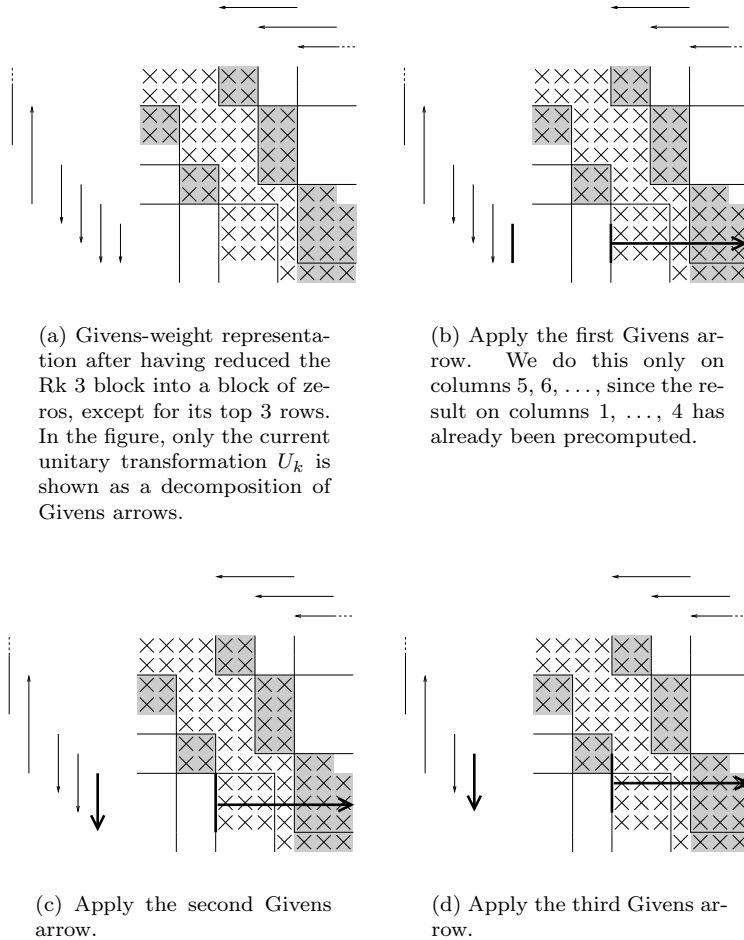


FIGURE 3.8. Preparative phase (a-d)

weights have turned from grey into white. The bottom right elements in Figure 3.8(a) are assumed to be disturbances coming from previous operations.

During the algorithm, we are faced with the following problem: the Givens-weight representation is by definition an *internal* representation, based on a QR-factorization, where the weights were stored each time just at the moment when they would go beyond the top border of the structure (Section 2). The problem is now that we want to apply these precomputed Givens transformations to the *whole* matrix, hereby also updating the representation in the upper triangular part.

Figures 3.8(b), 3.8(c) and 3.8(d) show the application of the Givens transformations belonging to the first three Givens arrows. Since part of their application has already been precomputed, these Givens arrows should be applied only to the columns lying on the right of their current action radius, in this case columns 5, 6, ... These operations are applied to the unstructured matrix part as well as to the weights of the upper triangular representation.

We should still explain why it is valid to apply the row operations directly to

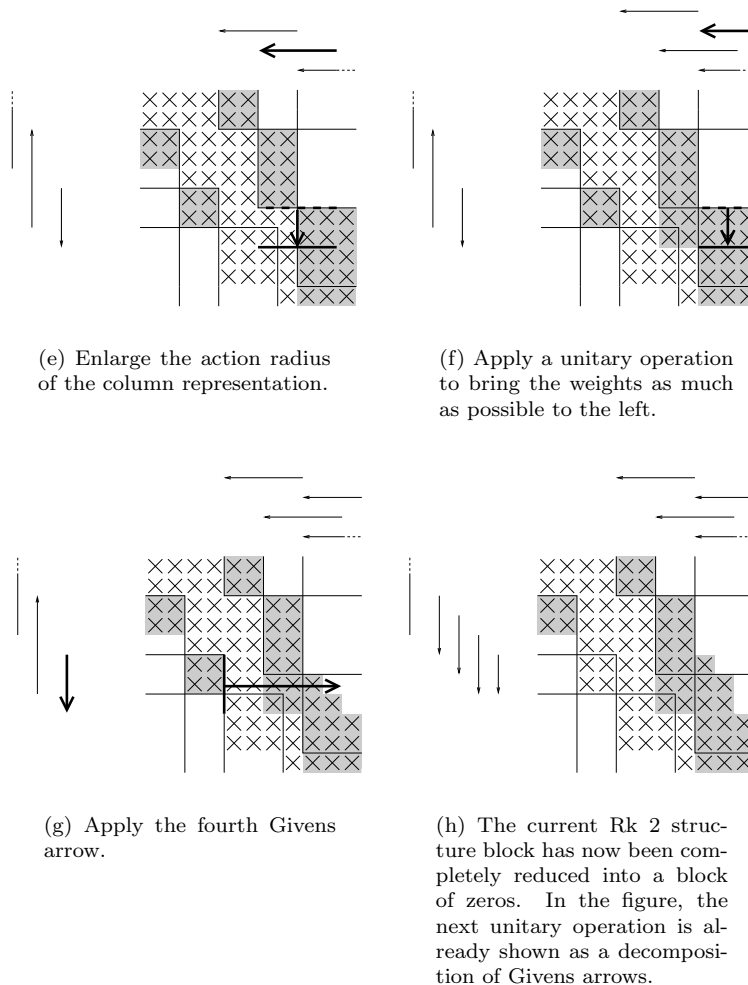


FIGURE 3.8. Preparative phase (e-h)

the weights of the upper triangular representation. To this end, we recall that the weights contain a kind of compressed information about the matrix, and that in order to obtain these elements in full form, the weights should first be spread out by the unitary column operations of the upper triangular Givens-weight representation. But clearly, by the associativity of matrix multiplication, it does not matter whether we first spread out the weights by the use of these column operations, or instead first apply the disturbing row operations. This shows that indeed, it is allowed to apply the row operations directly to the weights.

We are now at the point of applying also the fourth Givens arrow to the rows. But since we are going to ‘contaminate’ a new structure block in the upper triangular part, the application of this fourth Givens arrow would lead to a mix of real-size elements and weights, which is definitely not allowed.

The solution to this problem consists in ‘enlarging’ the column representation. This means that we bring the two rows lying just below the new structure block,

‘into’ the column representation: see Figure 3.8(e).

Having done this, it is now safe to apply the fourth Givens arrow. Before doing this, however, we note that applying all these row operations would ultimately lead to a complete fill-in in the upper triangular part. Since we want to minimize this fill-in as far as possible, we apply first an auxiliary unitary transformation to the columns, in order to bring the newly introduced weights as far as possible to the left: see Figure 3.8(f).

Having done all these preparations, we can finally apply the fourth Givens arrow to the rows: see Figure 3.8(g). We have then completely finished the Givens arrows belonging to the current unitary transformation U_k . Note that the corresponding weight block in Figure 3.8(h) has turned from grey into white: this indicates the fact that these elements do not contain ‘decoded’ information anymore, but that they contain precisely the real-size elements standing there at this particular moment of the algorithm. The reason underlying this, is nothing but the concept of Givens-weight representation.

At this moment, we are at the point of embarking the Givens arrows belonging to the second unitary operation. Because the situation in Figure 3.8(h) is similar to the one we started from in Figure 3.8(a), this process is not shown anymore.

Figure 3.9 summarizes the explained mechanism of the preparative phase in terms of the structure blocks in the structured upper triangular part.

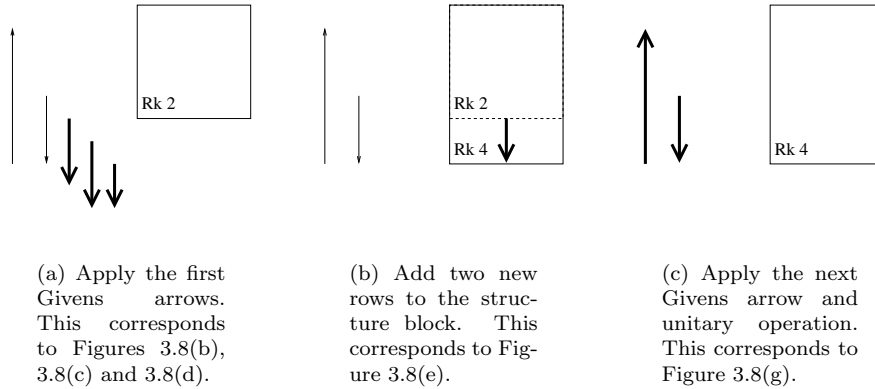


FIGURE 3.9. Mechanism underlying the structure block movement in the preparative phase.

Figure 3.10 shows the final situation for the example in Figure 3.8, at the end of the complete preparative phase. Note that the column representation has ‘grown’, corresponding to the fact that the ranks in the upper triangular part have increased, which is consistent with Figure 3.9.

It can also be seen from this figure that the representation for the block lower triangular part has completely *disappeared*. This underlies the fact that the structure blocks in the block lower triangular part have been transformed into blocks of zeros.

In order to make the matrix completely upper triangular, we should then still remove a few subdiagonals. The process of doing this, will be the subject of the next subsection.

3.3. Algorithm for the residual phase. In this subsection, we show that by using the Givens-weight representation, also the second phase of the QR-factorization

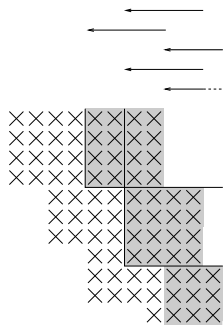


FIGURE 3.10. *Givens-weight representation after the complete preparative phase.*

can be performed in an efficient way, the so-called *residual phase*.

We will explain the algorithm for the Givens-weight representation shown in Figure 3.10.

The application of the residual phase makes use of the general techniques for updating the Givens-weight representation under the influence of Givens transformations reported in [4], in the form of what we called there a *generalized regression process*. This process is shown in Figure 3.11.

Let us comment on Figure 3.11. The basic flow of the algorithm is determined by applying the Givens arrows making the subsequent columns upper triangular. But we should be careful that there is no mixture of real-size elements and weights during this process. Therefore, before making a new column upper triangular, we first have to *spread out*, i.e. we first have to regress the action radius of the column representation if necessary. Figure 3.11(c) shows such an operation: we regress here from row 4 down to row 2, since we are intending to apply in the next step an operation acting on rows 3, ..., 7.

Note that these regression operations must proceed in the direction *opposite* to the original compression process. This means that when applying the inverse of the unitary operations highlighted in Figure 3.11(c), we must first apply the inverse of the auxiliary operations contained in the top right arrow, and only then can we apply the inverse of the original unitary operations contained in the bottom left arrow.

Having done these preparations, the next two columns are made upper triangular in Figures 3.11(d) and 3.11(e).

At this moment, we are at the point of embarking the following columns and making them upper triangular. Because this problem is similar to the one we started from in Figure 3.11(a), this process is not shown anymore.

Figure 3.12 summarizes the explained mechanism of the residual phase in terms of the structure blocks in the upper triangular part.

Figure 3.13 shows the final situation at the end of the complete residual phase. Note that the column representation has ‘lost’ some terrain, in the sense that it has regressed to the direction of the top right corner of the matrix. This is consistent with the mechanism illustrated in Figure 3.12. On the other hand, note that the unitary transformations involved in the Givens-weight representation for the block upper triangular part have remained exactly the same. This underlies the fact that the ranks and the induced column dependencies have been left unchanged under the regression process.

For a global overview, the reader could also have a second look at the original

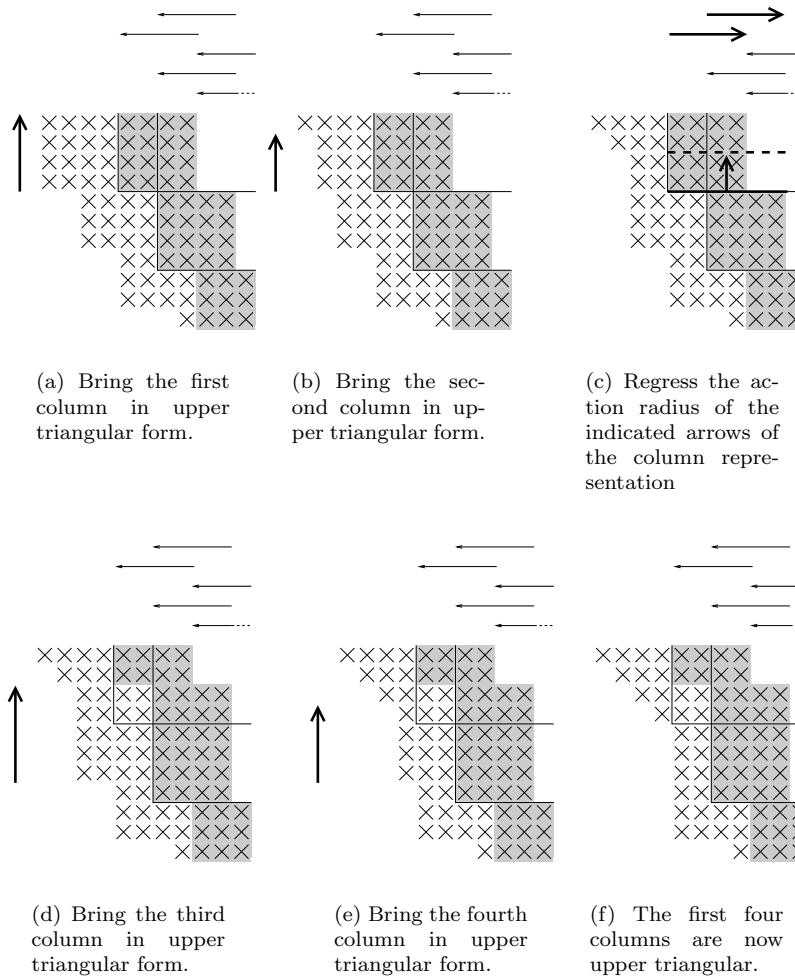


FIGURE 3.11. *Residual phase*

algorithm flow which was sketched in Figure 3.5.

Let us note that the above procedure for the residual phase guarantees that the structure blocks of R are all lying in the strictly upper triangular part of this matrix. In a certain sense, this may seem to conflict with Figure 3.3, which predicts that in certain cases, the structure blocks of R could reach beyond the main diagonal. In fact, the above algorithm will have performed an implicit *truncation* of structure in this case. But this did not occur for the example which we have chosen.

Summarizing, by the algorithm of the current section, we have obtained a QR-factorization $A = QR$, where the Q-factor is decomposed as a product of Givens transformations, and where the R-factor is represented by a column-based Givens-weight representation. We will now show how this QR-factorization can be used for the solution of a linear system.

4. Solution of a linear system. In this section we shall use the QR-factorization to solve a linear system $Ax = b$. We do this by rewriting the system in the form

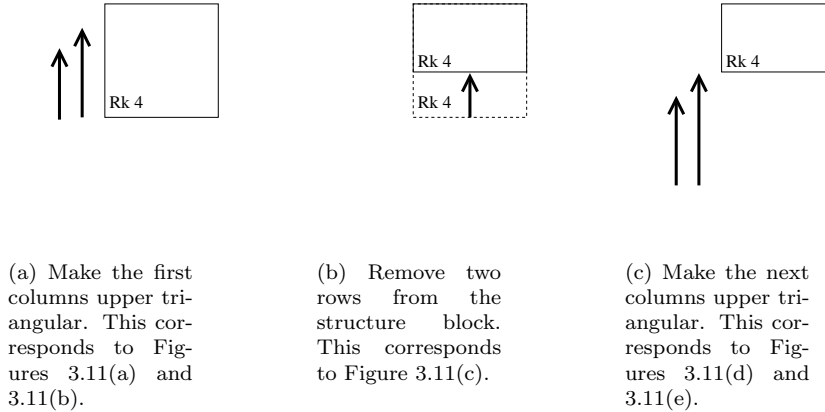


FIGURE 3.12. Mechanism underlying the structure block movement in the residual phase.

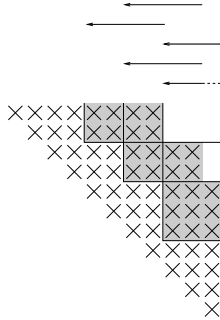


FIGURE 3.13. Givens-weight representation after the complete residual phase. The reader should check that the underlying structure blocks correspond with the predictions in Theorem 6.

$R\mathbf{x} = Q^H\mathbf{b}$, which can then be solved by backward substitution: see Figure 4.1.

Let us comment on this figure. The basic flow of the algorithm is determined by solving the subsequent rows of the upper triangular matrix R by backward substitution, hereby obtaining the subsequent components of the indeterminate vector \mathbf{x} , as in Figure 4.1(b). Note that in the latter figures, we used a column of crosses to denote the already computed components of \mathbf{x} .

We are then at the point of ‘entering’ a new structure block in the upper triangular part of the R-factor. This is the right moment to multiply the matrix R by the precomputed unitary column operation associated to this structure block, hereby compressing the matrix. Although this compression may sound rather expensive, from the computational point of view, nothing has to be done since the effect of this unitary column operation has already been *precomputed*, by the concept of Givens-weight representation. (In principle we should still apply this operation to the rows below the current action radius, but these rows have already been solved and thrown away). The only operation that actually *has* to be performed, is to multiply the indeterminate vector \mathbf{x} with the inverse of these unitary column operations. (In order not to overwrite the already computed values, one should use an auxiliary vector for performing these operations on.) The latter operations are indicated by the fat

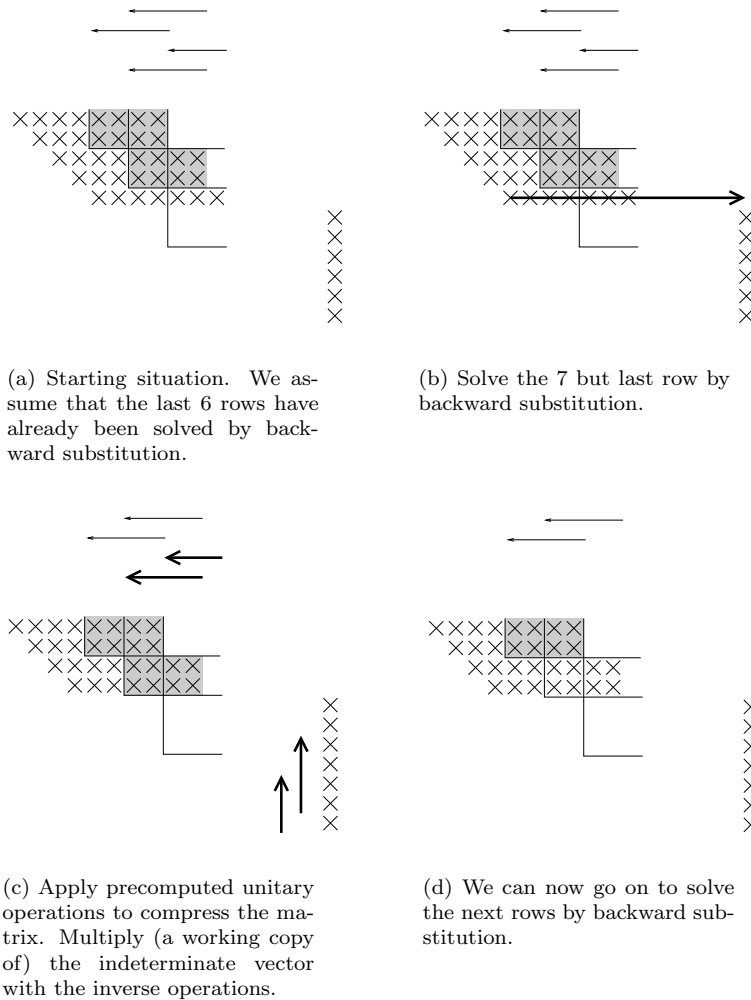


FIGURE 4.1. *Direct solution of a linear system*

vertical arrows in Figure 4.1(c).

Let us point out that, since these vertical arrows contain the inverse operations of the horizontal arrows in Figure 4.1(c), one could expect them to point in the direction *opposite* to the one indicated in Figure 4.1(c); but this is not correct since the former arrows act on columns while the latter act on the *rows*.

We can then go on to compute the next components of the indeterminate vector by backward substitution, until a new structure block is reached. Since the situation in Figure 4.1(d) is similar to the one we started from in Figure 4.1(a), these operations are not shown anymore.

At the end of this process we will have obtained the full indeterminate vector \mathbf{x} , hereby solving the linear system.

For completeness of this paper, let us now describe a similar solution algorithm in case where the R-factor is described by a *row-based* representation. Actually, we

prefer to explain the algorithm in terms of a *lower triangular* matrix L (which is an equivalent problem since we could rewrite $R\mathbf{x} = \mathbf{b}$ as $LJ\mathbf{x} = J\mathbf{b}$, where $L := JRJ$, and with J the antidiagonal matrix. The row-based Givens-weight representation for R transforms in this way into a row-based Givens-weight representation for the matrix L .)

The algorithm is explained in Figure 4.2.

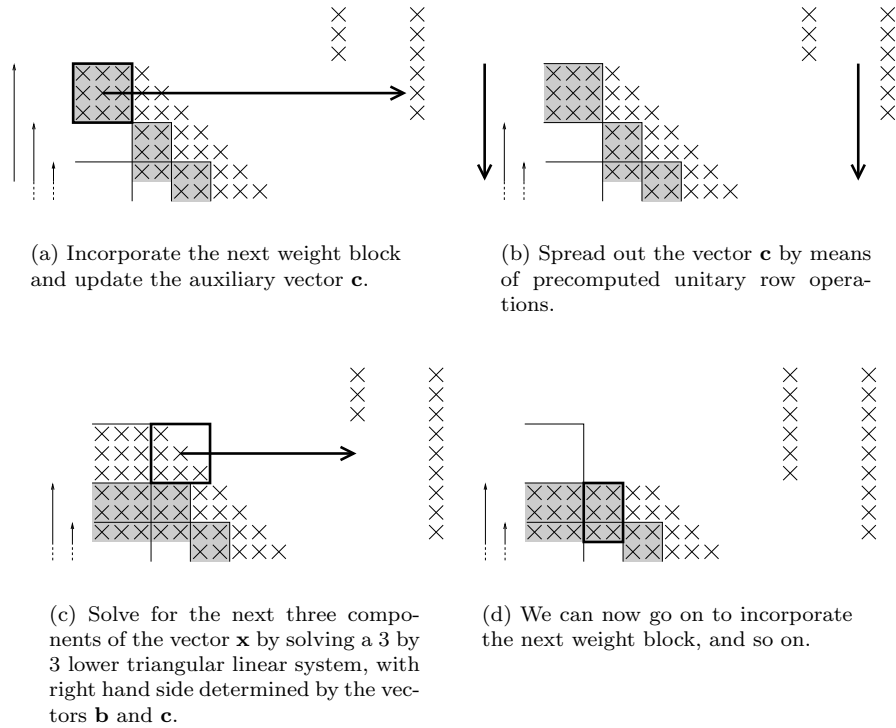


FIGURE 4.2. Direct solution of a linear system in case of a row-based Givens-weight representation

Let us comment on this figure. Figure 4.2(a) shows the starting situation, where it is assumed that the first three rows have already been solved by forward substitution, hereby yielding the three components of the indeterminate vector \mathbf{x} shown with crosses on top of the figure. Note that we have also a second vector \mathbf{c} standing in the figure, of which already six components have been computed. This will be an auxiliary vector. It contains the matrix-vector product of the already computed components of \mathbf{x} with the structured lower triangular matrix part.

Figure 4.2(a) shows how to update this auxiliary vector \mathbf{c} by incorporating the next weight block and multiplying it with the three last computed components of \mathbf{x} . This contribution is then added to the vector \mathbf{c} .

Figure 4.2(b) shows how we spread out the weight matrix. These operations serve only for understanding the algorithm, but they are not actually computed. What we do perform, is spreading out the vector \mathbf{c} by means of these same operations.

Since the top three rows in Figure 4.2(c) have been completely spread out to their actual form, it is now safe to solve for the next three components of \mathbf{x} . The coefficient matrix of this linear system is given by the lower triangular 3 by 3 matrix surrounded

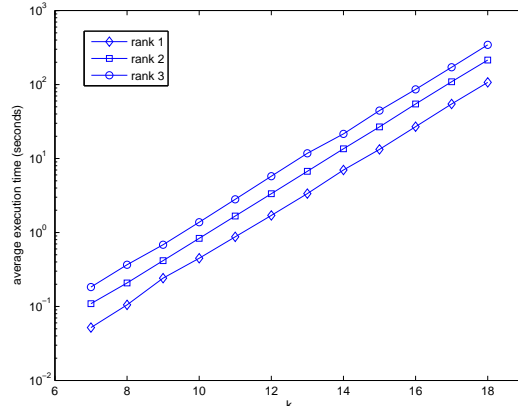


FIGURE 5.1. Average execution time for 5 random samples of size $n = 2^k$ and rank $r = 1, 2, 3$

by the thick black box in Figure 4.2(c). The right hand side of the linear system is determined by the actual right hand side vector \mathbf{b} , from which is subtracted the contribution of the matrix-vector product of the already computed components of \mathbf{x} with the structured lower triangular matrix part, which is contained in the vector \mathbf{c} .

We can then move on to the next weight block in Figure 4.2(d). The next operations are not shown anymore.

We note that the flow of this row-based algorithm is very similar to that for block quasiseparable matrices, first reported in [9]. On the other hand, the column-based algorithm described earlier in this section does not seem to have such an interpretation.

REMARK 7. (*Overdetermined systems:*) *In this and the previous section, we implicitly assumed that the coefficient matrix A , and hence the R -factor of its QR-factorization were square matrices. But this condition is irrelevant: also in the case of a (full-rank) overdetermined linear system with $A \in \mathbb{C}^{m \times n}$ and $m \geq n$, one can compute the QR-factorization and the corresponding least-squares solution to a linear system in exactly the same way as before.*

5. Numerical experiments. To check the accuracy and the numerical stability of the algorithms to compute the QR-factorization and solve the corresponding linear system based on this factorization, we have performed several numerical experiments. The algorithms were implemented in Matlab*. The experiments were executed on an Intel PC running Matlab Version 7.0.1.24704 (R14) under Linux having 1GByte of memory and an Intel Pentium 4 processor running at 3.2 GHz. The software of these numerical experiments can be requested from the authors.

Experiment 1: We constructed non-symmetric rank structured matrices of sizes $n \times n$, with $n = 2^k$ for $k = 7, \dots, 18$. The structure blocks were situated just below, and just above the main diagonal. For each size, the rank indices $r = 1, 2, 3$ were taken. For each of these sizes n and each of these rank indices r , 5 samples were considered. Figure 5.1 shows for each size $n = 2^k$ and each rank index r the execution time $T_{k,r}$ averaged over the 5 samples of computing the QR-factorization and solving the corresponding linear system.

To check that the computational complexity is linear in the size n of the matrix, Figure 5.2 shows the fraction $T_{k+1,r}/T_{k,r}$ averaged over the 5 samples and over the

*Matlab is a registered trademark of The MathWorks, Inc.

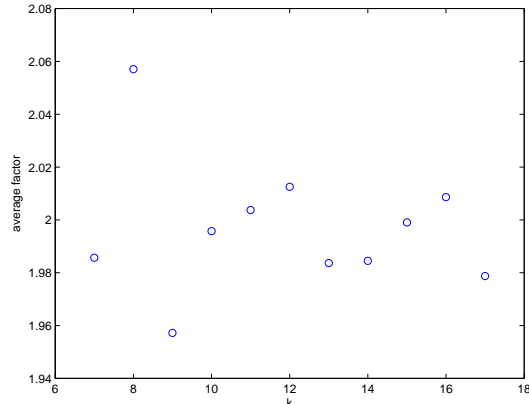


FIGURE 5.2. Fraction $T_{k+1,r}/T_{k,r}$ averaged over 5 random samples and over ranks $r = 1, 2, 3$ in function of size $n = 2^k$

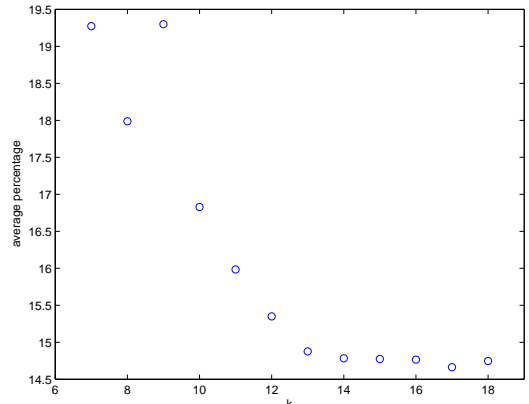


FIGURE 5.3. Percentage of total computing time spent to solve the linear system averaged over 5 random samples and over ranks $r = 1, 2, 3$ in function of size $n = 2^k$

ranks $r = 1, 2, 3$.

Figure 5.3 gives the average percentage of the total execution time spent to solve the linear system once the QR-factorization is computed.

To check the accuracy of the algorithm, we considered matrices having singular values equidistant between 10^{-1} and 1, i.e., the condition number of each of the 5 samples is equal to 10. In the same way we took for each size n and each rank r 5 samples having condition number 10^{10} . To measure the accuracy, we computed the relative residual norm

$$\frac{\|Ax - b\|}{\|A\|\|x\| + \|b\|}.$$

Figure 5.4 shows the relative residual norm averaged over the 5 samples and the rank $r = 1, 2, 3$ for each of the condition numbers and each of the sizes.

Finally, let us give somewhat more details on the construction of the above test matrices. Starting from a diagonal matrix containing the desired singular values, we applied to rows and columns a ‘disturbing’ sequence of Givens arrows of width r . This

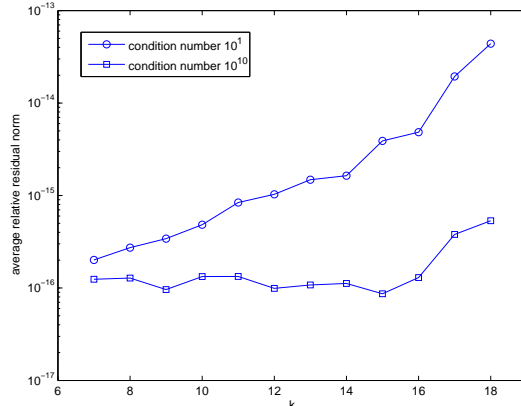


FIGURE 5.4. Relative residual norm averaged over 5 random samples and over ranks $r = 1, 2, 3$ in function of size $n = 2^k$ and condition number 10^1 and 10^{10}

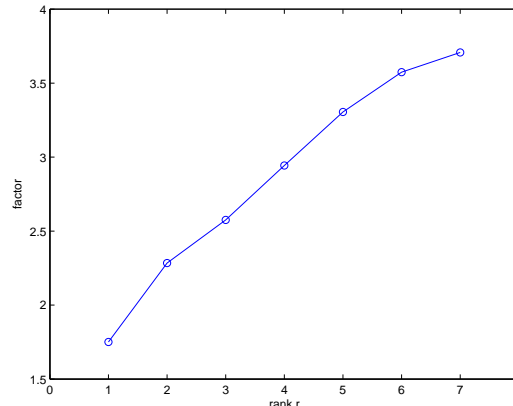


FIGURE 5.5. Fraction T_{2r}/T_r in function of the rank index $r = 2^l$ with $l = 1, 2, \dots, 7$.

resulted in a non-symmetric matrix having the required rank structure in both lower and upper triangular part.

Since it can be argued that the above construction yields rather ‘special’ rank structured matrices, we next applied a ‘randomization’ procedure. We did this by applying Givens transformations to rows and columns, in such a way that both the lower and the upper rank structure of the matrix was preserved. Let us point out that this randomization procedure took about 95% of our total execution time. A detailed description of this perturbation method will not be given here.

Experiment 2: To check the computational complexity as a function of the rank index r , we considered the execution time T_r for matrices of fixed size $n = 2^{10} = 1024$ and varying rank index $r = 2^l$ with $l = 1, 2, \dots, 8$. The actual construction of the test matrices was performed in exactly the same way as before. Figure 5.5 gives the fraction T_{2r}/T_r for subsequent ranks. Note that the fraction tends to approximate 4 for large rank indices r but is much smaller for small values of r .

Experiment 3: To indicate that our implementation works for arbitrary rank structures, we considered one sample of size $n = 2^k$, with $k = 7, 8, \dots, 18$ having a rank structure as follows. The distance between two subsequent structure blocks

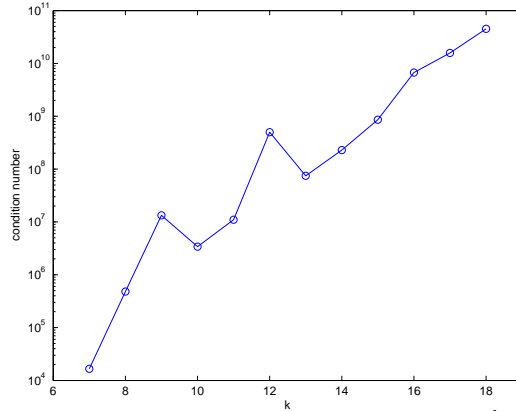


FIGURE 5.6. Condition number in function of the size $n = 2^k$ of the matrix

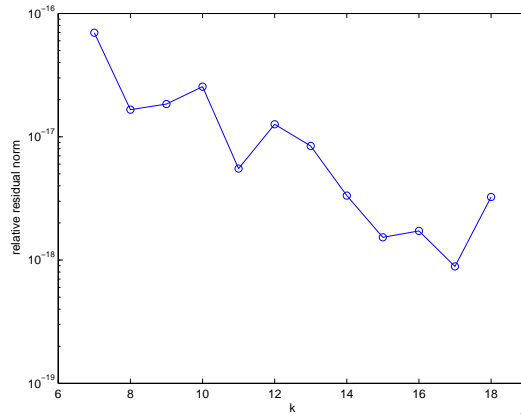


FIGURE 5.7. Relative residual norm in function of the size $n = 2^k$ of the matrix

was equal to 2, and both lower and upper structure blocks were lying at a distance 2 from the main diagonal. The average rank index was about $\log n - 2 = k - 2$, with rank indices being systematically decreased when the structure blocks approached the border of the matrix. For example, the lower structure block coordinates for $n = 2^5 = 32$ were given by

$$\begin{bmatrix} i_k \\ j_k \\ r_k \end{bmatrix} = \begin{bmatrix} 5 & 7 & 9 & 11 & 13 & 15 & 17 & 19 & 21 & 23 & 25 & 27 & 29 \\ 2 & 4 & 6 & 8 & 10 & 12 & 14 & 16 & 18 & 20 & 22 & 24 & 26 \\ 1 & 2 & 2 & 3 & 3 & 3 & 3 & 3 & 3 & 3 & 3 & 2 & 2 & 1 \end{bmatrix}.$$

Although the underlying rank structure was chosen to be symmetric in lower and upper triangular parts, the matrices themselves were non-symmetric.

Figure 5.6 shows the condition number of the constructed matrix in function of the size $n = 2^k$ while Figure 5.7 plots the relative residual norm.

Figure 5.8 shows the execution time T_k in function of the size while Figure 5.9 plots the fraction between subsequent execution times. We note that the last sample required a higher computation time since the storage of the Givens-weight representation was close to the memory limits.

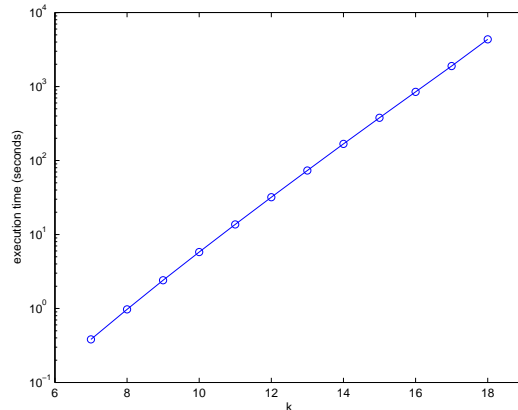


FIGURE 5.8. Execution time T_k in function of size $n = 2^k$ for $k = 7, 8, \dots, 18$

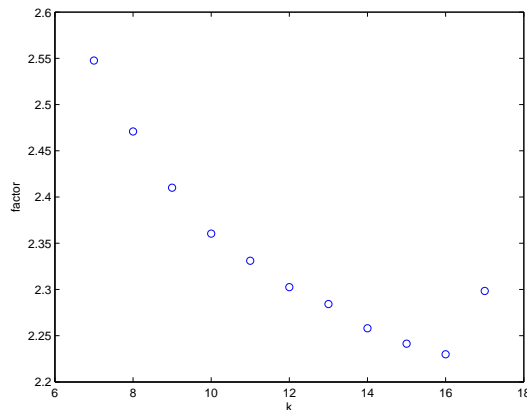


FIGURE 5.9. Fraction T_{k+1}/T_k in function of size $n = 2^k$ for $k = 7, 8, \dots, 17$

6. Conclusion. In this paper we described an algorithm to perform the QR-factorization of a rank structured matrix, using the Givens-weight representation. We showed how this QR-factorization could be used as a first step for solving linear systems. We described the underlying propagation of rank structure during the algorithm. The numerical performance of the algorithm was demonstrated by means of some numerical experiments.

REFERENCES

- [1] S. Chandrasekaran, P. Dewilde, M. Gu, T. Pals, and A.-J. van der Veen. Fast stable solver for sequentially semi-separable linear systems of equations. *Lecture Notes in Computer Science*, 2552:545–554, 2002.
- [2] S. Chandrasekaran and M. Gu. Fast and stable algorithms for banded plus semiseparable systems of linear equations. *SIAM Journal on Matrix Analysis and its Applications*, 25(2):373–384, 2003.
- [3] S. Delvaux and M. Van Barel. Structures preserved by the QR-algorithm. *Journal of Computational and Applied Mathematics*, 187(1):29–40, 2005. DOI 10.1016/j.cam.2005.03.028.
- [4] S. Delvaux and M. Van Barel. A Givens-weight representation for rank structured matrices. Report TW 453, Department of Computer Science, Katholieke Universiteit Leuven, Leuven, Belgium, March 2006.

- [5] S. Delvaux and M. Van Barel. Rank structures preserved by the QR-algorithm: the singular case. *Journal of Computational and Applied Mathematics*, 189:157–178, 2006.
- [6] P. Dewilde and A.-J. van der Veen. *Time-varying systems and computations*. Kluwer Academic Publishers, Boston, June 1998.
- [7] Y. Eidelman and I. C. Gohberg. On a new class of structured matrices. *Integral Equations and Operator Theory*, 34:293–324, 1999.
- [8] Y. Eidelman and I. C. Gohberg. Fast inversion algorithms for a class of block structured matrices. *Contemporary Mathematics*, 281:17–38, 2001.
- [9] Y. Eidelman and I. C. Gohberg. A modification of the Dewilde-van der Veen method for inversion of finite structured matrices. *Linear Algebra and its Applications*, 343-344:419–450, April 2002.
- [10] Y. Eidelman and I. C. Gohberg. Fast inversion algorithms for a class of structured operator matrices. *Linear Algebra and its Applications*, 371:153–190, 2003.
- [11] I. C. Gohberg, T. Kailath, and I. Koltracht. Linear complexity algorithms for semiseparable matrices. *Integral Equations and Operator Theory*, 8(6):780–804, 1985.
- [12] E. Van Camp, N. Mastronardi, and M. Van Barel. Two fast algorithms for solving diagonal-plus-semiseparable linear systems. *Journal of Computational and Applied Mathematics*, 164-165:731–747, 2004.
- [13] R. Vandebril, M. Van Barel, and N. Mastronardi. A note on the representation and definition of semiseparable matrices. *Numerical Linear Algebra with Applications*, 12(8):839–858, October 2005.